On Effective Classification of Strings with Wavelets

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

ABSTRACT

In recent years, the technological advances in mapping genes have made it increasingly easy to store and use a wide variety of biological data. Such data are usually in the form of very long strings for which it is difficult to determine the most relevant features for a classification task. For example, a typical DNA string may be millions of characters long, and there may be thousands of such strings in a database. In many cases, the classification behavior of the data may be hidden in the compositional behavior of certain segments of the string which cannot be easily determined apriori. Another problem which complicates the classification task is that in some cases the classification behavior is reflected in global behavior of the string, whereas in others it is reflected in local patterns. Given the enormous variation in the behavior of the strings over different data sets, it is useful to develop an approach which is sensitive to both the global and local behavior of the strings for the purpose of classification. For this purpose, we will exploit the multi-resolution property of wavelet decomposition in order to create a scheme which can mine classification characteristics at different levels of granularity. The resulting scheme turns out to be very effective in practice on a wide range of problems.

1. INTRODUCTION

In recent years, it has become increasingly easy to store and record a wide variety of string data for a number of applications. Examples of such data include proteins which often contain long sequences of amino acids. Another class of data which are closely related to strings are time series or sequential data in which sequences of events are stored in strings [11]. A number of approaches for traditional problems such as clustering, indexing and subpattern identification have also been developed for this domain [3, 9, 10, 13].

An important data mining problem is that of classification. The classification problem has been widely studied in the data mining, artificial intelligence and machine learning communities and is defined as follows: we have a set of records called the *training data*, in which each record is labeled with a *class*. This training data is used to construct a model which relates the features in the data records to the class label. For a given record for which the class label is unknown, this model may be used to predict its class label. This problem often arises in the context of customer profiling, target marketing, medical diagnosis, and speech recognition. Examples of techniques which are often used for classification in the data mining domain include decision trees, rule based classifiers, nearest neighbor techniques and neural networks [5, 6, 7, 8]. A detailed survey of classification methods may be found in [8].

The string domain provides some interesting applications of the classification problem. An important example is the biological domain in which large amounts of data have become available in the last few years. Applications of DNA matching and identification include the fields of archeology, forensics and medicine [5]. Other examples include sequential data for event classification, and classification of customer data for user profiling. In many of these cases, the resulting strings are quite long and vary from a few hundred symbols to the thousands. For applications in which the strings are very long, the classification problem turns out to be very perplexing in pratice. In most cases, both the global and local composition of the proteins may influence its classification behavior. For example, a typical protein sequence may contain thousands of amino acids, which are drawn from the fixed alphabet $\Sigma = \{\sigma_1 \ldots \sigma_l\} = \{A, C, T, G\}$. In most cases, the *compositional behavior* of the sequence in certain subportions may significantly affect the physical characteristics of the corresponding protein. In other cases, certain kinds of proteins may show local or periodic presence of different kinds of amino acids. These characteristics may be hard to distinguish at the global level and significantly complicate the classification process. Since each sequence may contain thousands of characters, it is also a difficult and complex problem of finding the most discriminatory compositions at the correct level of detail and granularity. This is true for a large number of applications in which the classification behavior of the relevant strings can only be accurately determined by taking *both* the compositional and positional behavior of the constituents into account.

The preferred method for string classification is that of the nearest neighbor technique based on the edit distance [4]. In this method, the class label of the nearest neighbor to the test instance is reported as the relevant label. This technique has several drawbacks: (1) The edit distance turns out to be

SIGKDD '02 Edmonton, Alberta, Canada

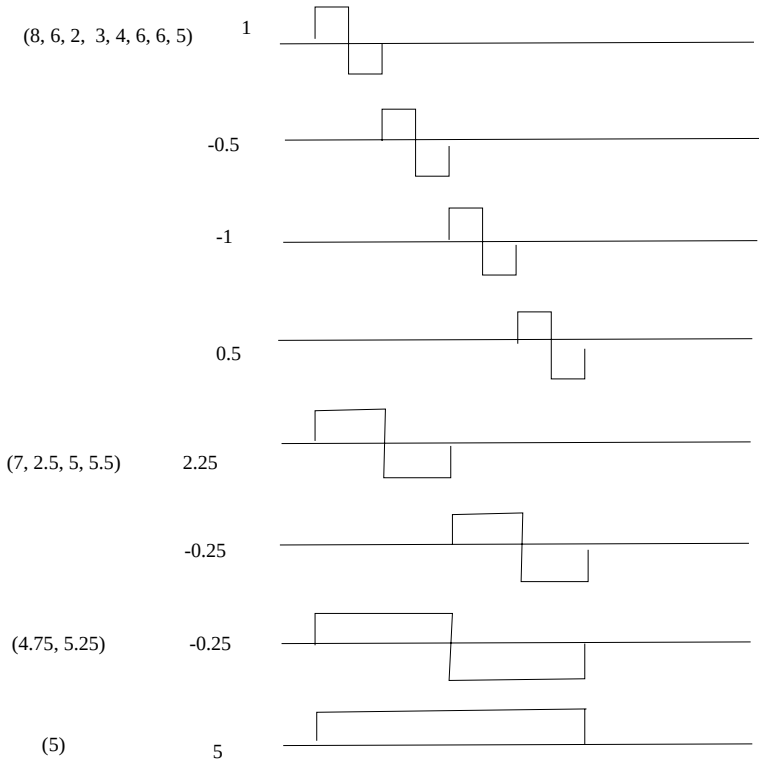

Figure 1: Illustration of the Wavelet Decomposition

difficult to compute in practice. Furthermore, the system is difficult to implement efficiently since a function such as the edit distance is not effectively indexable. This restricts the applicability of the method for very large scale applications. (2) Locality in the feature space for long strings of variable length is rarely a good measurement of locality in class behavior. For example, two strings of very different lengths may have similar classification characteristics because of certain important local combinations of symbols. However, the edit distance between such strings is likely to be at least as large as the the difference in their length. Therefore, it is useful to create a technique which can mine such local combinations of patterns effectively. Also, in many biological applications, only sample string fragments are available rather than the entire strings. In such cases, global characteristics tend to be an especially poor indicator of the classification behavior. (3) The distance based classifier has an overwhelming dependence on the exact positioning of the alphabets in the strings. This dependence results in an inability to capture important *compositional characteristics* of the strings. For example, in a biological application, certain broad classes of proteins may have similar composition but may often vary considerably in the exact ordering of the characters depending upon the particular subclass that the protein may belong to. Such behavior cannot be captured well by distance based classifiers.

The wavelet decomposition technique has recently been recognized as a useful tool for a number of database applications. A comprehensive overview of the wavelet technique may be found in [17]. An important property of the wavelet technique is that it creates a hierarchical decomposition of the data which can capture trends at varying levels of granularity. As we shall see in this paper, this creates a string classification system which uses important and non-obvious discriminatory characteristics in the classification process. While the wavelet technique helps considerably in the creation of a system which provides multi-resolution insight into the data characteristics, we combine it with a rule based technique in order to make the classifier sensitive to particular local characteristics of the strings.

This paper is organized as follows. In the remainder of this section, we will discuss the contributions of this paper, notations and background of the wavelet decomposition technique. In section 2, we will discuss how to use this technique in order to build an effective rule based classifier for the data. In section 3, we will discuss the empirical results. Section 4 contains the conclusions and summary.

1.1 Contributions of this Paper

This paper discusses an effective wavelet based technique for string classification. This wavelet based approach provides the ability to isolate the most discriminatory compositions of the strings at varying levels of analysis. In addition, a rule based approach to the problem ensures that the local compositional characteristics of the strings are used during the classification process. The result is a system which is not only much more effective than the currently used nearest neighbor classifier, but is also significantly more efficient during the classification process.

1.2 Background and Notations

In order to facilitate further development of the ideas in

Table 1: An Example of Wavelet Coefficient Computation

Granularity (Order k)	Averages Φ values	DWT Coefficients ψ values
$k = 4$	(8, 6, 2, 3, 4, 6, 6, 5)	-
$k = 3$	(7, 2.5, 5, 5.5)	(1, -0.5,-1, 0.5)
$k = 2$	(4.75, 5.25)	(2.25, -0.25)
$k = 1$	(5)	(-0.25)

this paper, we will introduce some notations and definitions. We assume that the training data set $\mathcal{D}$ contains N strings, such that the length of the ith string is denoted by d_i. We assume that each of the N strings is drawn from the alphabet $\Sigma = \{\sigma_1 \ldots \sigma_l\}$. We also assume that associated with each record in the data set $\mathcal{D}$, we have one of a set of k class labels drawn from $C_1 \ldots C_k$. The classification model is constructed using the records in the database $\mathcal{D}$ along with their corresponding class labels.

1.3 Wavelet Decomposition and Modifications for String Domain

In the method discussed in this paper, we will utilize a method which we refer to as the *Haar Wavelet*. We will also discuss a brief algorithmic overview of the method used in order to generate the Haar coefficients from a given record and the modifications necessary for their use in a string domain containing sequences which are drawn from a fixed alphabet.

The basic idea in the wavelet technique is to create a decomposition of the data characteristics into a set of wavelet functions and basis functions. The property of the wavelet method is that the higher order coefficients of the decomposition illustrate the broad trends in the data, whereas the more localized trends are captured by the lower order coefficients.

We assume for ease in description that the length q of the series is a power of 2. The Haar Wavelet decomposition defines 2^{k-1} coefficients of order k. Each of these 2^{k-1} coefficients corresponds to a contiguous portion of the time series of length $q/2^{k-1}$. The ith of these 2^{k-1} coefficients corresponds to the segment in the series starting from position $(i-1) \cdot q/2^{k-1} + 1$ to position $i * q/2^{k-1}$. Let us denote this coefficient by ψ_k^i and the corresponding time series segment by S_k^i. At the same time, let us define the average value of the first half of the S_k^i by a_k^i and the second half by b_k^i. Then, the value of ψ_k^i is given by $(a_k^i - b_k^i)/2$. More formally, if Φ_k^i denote the average value of the S_k^i, then the value of ψ_k^i can be defined recursively as follows:

$$\psi_k^i = (\Phi_{k+1}^{2 \cdot i-1} - \Phi_{k+1}^{2 \cdot i})/2 \tag{1}$$

The set of Haar coefficients is defined by the Ψ_k^i coefficients of order 1 to $\log_2(q)$. In addition, the global average Φ_1^1 is required for the purpose of perfect reconstruction. We note that the coefficients of different order provide an understanding of the major trends in the data at a particular level of granularity. For example, the coefficient ψ_k^i is half the quantity by which the first half of the segment S_k^i is larger than the second half of the same segment. Since larger values of k correspond to geometrically reducing segment sizes, one can obtain an understanding of the basic trends at different levels of granularity.

We note that this definition of the Haar wavelet makes it very easy to compute by a sequence of averaging and differencing operations. In Table 1, we have illustrated how the wavelet coefficients are computed for the case of the sequence $(8, 6, 2, 3, 4, 6, 6, 5)$. This decomposition is illustrated in graphical form in Figure 1.

We note that the definition discussed above is for the case of a quantitative time series. In order to extend this method to strings defined over the discrete alphabet $\Sigma = \{\sigma_1 \ldots \sigma_l\}$, we associate a set of l coefficients for each region in the hierarchical decomposition. Therefore, analogous to the values of Φ_k^i and ψ_k^i in the decomposition, we define the coefficients $\Phi_k^i(j)$ and $\psi_k^i(j)$ for $j \in \{1 \ldots l\}$. Here the value of $\Phi_k^i(j)$ is defined as the *fraction* of positions in S_k^j which contain the symbol σ_j. Therefore, we have:

$$\sum_{j=1}^{l} \Phi_k^i(j) = 1 \tag{2}$$

Correspondingly, the value of $\psi_k^i(j)$ is defined as follows:

$$\psi_k^i(j) = (\Phi_{k+1}^{2 \cdot i-1}(j) - \Phi_{k+1}^{2 \cdot i}(j))/2 \tag{3}$$

We note that the value of $\psi_k^i(j)$ is half the difference in the fraction of the presence of the symbol σ_j in the first and second half of segment S_k^i. We also note two properties of the wavelet decomposition:
(1) The number of wavelet coefficients is l times the size of the string. For each string symbol we have as many coefficients as the length of the string.
(2) Each wavelet coefficient lies between -1 and $+1$. Note that a given position can be interpreted in terms of the differences in fractional compositions of two intervals.

2. THE WAVELET CLASSIFICATION TECHNIQUE

The aim of the wavelet representation is to develop a technique which is sensitive to the some of the difficulties in performing effective classification in string applications. For example, two strings in a biological domain may show similar classification not just on the position of the characters, but also in their compositional behavior. At the same time, we do not want to completely ignore the ordering information since it may be relevant to the classification.

Since lower order features in the wavelet decomposition encode the global composition of long segments of the string, it is possible to characterize the global compositional behavior of large substrings by using a combination of a small number of features. In addition, by using ordered combinations of higher order wavelet coefficients, it is possible to find very local sets of patterns which are highly indicative

Algorithm *WavRules(Training Database:*$\mathcal{D}$;
minimum support: s, minimum confidence: c,
Maximum Gap: maxgap);
begin
$\mathcal{W}' = CreateWavRepresentation(\mathcal{D})$;
$\mathcal{R} = CreateRules(\mathcal{W}', s, c,$ maxgap);
$(\mathcal{R}', defaultclass) = ReorderRules(\mathcal{R}, \mathcal{W}')$;
end

Figure 2: The Training Phase of the Wavelet Classifier

Algorithm *CreateWavRepresentation(Database:* $\mathcal{D}$*)*;
begin
$\mathcal{W}' = \{\}$;
for each string in $\mathcal{D}_F$ determine wavelet coefficients
as described in subsection 1.3;
Create a set of ϕ ranges covering values from -0.5 to $+0.5$;
{ Thus, the ith range corresponds to values between
$-0.5 + (i-1)/\phi$ and $-0.5 + i/\phi$ for $i \in \{1, \ldots \phi\}$ };
for each coefficient in each string in $\mathcal{D}$, replace
it with the interval in which it lies and
add the resulting string to $\mathcal{W}'$;
return($\mathcal{W}'$);
end

Figure 3: Determining the Wavelet Representation

of class behavior. In this section, we will discuss a classifier which builds on these representational advantages of the wavelet technique.

The overall training phase is illustrated in Figure 2. We assume that the input to the algorithm is the set of N strings in the database $\mathcal{D}$, a minimum support s, a minimum confidence c, and a parameter called *maxgap* which quantifies non-relevant lengths of strings in the classification process. We will discuss these parameters in more detail slightly later.

The training process of the classifier works in three phases. (1) In the first phase, the wavelet representation of the data is constructed. This is discretized in order to create a binary representation which is necessary for an effective rule based classifier. This process is denoted by the subroutine *CreateWavRepresentation* in Figure 2. (2) In the second phase we determine the set of *ordered compositional rules* which are most indicative of class behavior. The beauty of compositional rules is that they leverage the flexibility of the wavelet decomposition process in order to identify combinations of both composition as well as the ordering behavior of the segments in order to model the class discrimination in the data. Such ordered compositional behavior identifies a local region of the string in which the corresponding trends occur. The length of this local region is heavily dependent on the detail level of the corresponding wavelet coefficients. This essentially defines the level of granularity of the corresponding classification rules. This process is denoted by the procedure *CreateRules* in Figure 2. (3) Once these compositional rules have been constructed, we prune them in order to reduce overfitting and improve the effectiveness of the classification model. This procedure is denoted by *ReorderRules* in Figure 2. These rules are then used in order to classify individual test instances. In the next subsections,

Algorithm *CreateRules(Wavelet Transformed Database:* $\mathcal{W}'$,
Minimum Support: s Minimum Confidence: c,
Maximum Gap: maxgap);
begin
for $gap = 0$ to $maxgap$ **do**
begin
Determine all 2-patterns which have interval gap
between them, and which satisfy the required
minimum support s;
{ We denote this set by $\mathcal{L}_2(gap)$};
end;
$\mathcal{C}_2 = \cup_{gap=0}^{maxgap} \mathcal{L}_2(gap)$;
$k = 2$;
while $\mathcal{L}_k$ is not null **do**;
begin
Determine $\mathcal{C}_{k+1}$ by performing self join on $\mathcal{L}_k$;
Use database $\mathcal{W}'$ to find support counts of each
candidate in $\mathcal{C}_{k+1}$;
Assign $\mathcal{L}_{k+1}$ as the set of candidates with support
greater than s;
$k = k + 1$;
end
$\mathcal{L} = \cup_{i=1}^{(k+1)} \mathcal{L}_i$;
for each pattern in $P \in \mathcal{L}$ and class C_i, generate the
rule $P \Rightarrow C_i$ if the corresponding rule has
minimum confidence c;
{ We note that the class labels of records in database $\mathcal{W}'$
are needed for calculation of the confidence of patterns; }
{ We denote the corresponding rule set by $\mathcal{R}$; }
return($\mathcal{R}$);
end

Figure 4: Effective Creation of the Wavelet Rules

Algorithm *TruncateRules(Original Rule Set:* $\mathcal{R}$,
Transformed Database: $\mathcal{W}'$*)*;
begin
Order rules in $\mathcal{R}$ in sequence of decreasing confidence;
for each data point $x \in \mathcal{W}'$ find highest
precedence rule covered by it;
if consequent of rule is same as class label of x **then**
mark rule and delete x from $\mathcal{W}'$;
$\mathcal{R}'$ = Set of all marked rules in $\mathcal{R}$;
{ The rules in $\mathcal{R}'$ are stored in order of precedence; }
$defaultclass$ = Majority Class from remaining data $\mathcal{W}'$;
return($\mathcal{R}'$, $defaultclass$);
end

Figure 5: Effective Pruning of the Wavelet Rules

Algorithm *Classify(TestInstance: T, Rules: R', Default Class: defaultclass)*;
begin
Transform test instance T into wavelet representation T_W;
Find highest precedence rule for which T_W is a subpattern of the antecedent of the rule R;
if no such class exists return(*defaultclass*);
else return class in consequent of R;
end

Figure 6: The classification procedure for the wavelet method

we will describe the details of each of these phases.

2.1 Creation of the Wavelet Representation

This procedure is denoted by *CreateWavRepresentation* and is described in Figure 3. In the first step, we determine the wavelet coefficients of the strings in $\mathcal{D}$. We note that wavelet coefficients of different orders correspond to intervals which may possibly subsume one another. Each of these coefficients is then discretized into ϕ intervals. Specifically, we create a set of ϕ ranges covering the values from -0.5 to $+0.5$. Thus, the ith range corresponds to values between $-0.5 + (i-1)/\phi$ and $-0.5 + i/\phi$. The reason for the use of this range is that each wavelet coefficient is half the difference of the fractional composition of the current interval with that of an adjacent interval. Thus, all coefficients lie in the range $(-0.5, 0.5)$. The only exception are the l wavelet coefficients corresponding to $\Phi_1^1(j)$ which are the global averages across the decomposition. These coefficients are separately discretized into ϕ intervals between 0 and 1. We shall refer to the resulting string as the *discretized wavelet decomposition.*

2.2 Creation of the Rules for the Wavelet Representation

In this section, we will discuss the process of rule construction for the wavelet representation. While the wavelet representation provides an overview of the variations in compositional characteristics at different levels of granularity, it is useful to isolate localized regions in the data where discriminatory variations in compositional behavior occur. For this purpose, we will define the concept of a *compositional pattern*:

DEFINITION 2.1. *A compositional pattern $O < v_1, id_1 > g_1 < v_2, id_2 > g_2 \ldots g_{k-1} < v_k, id_k >$ is an alternating sequence of discretized coefficient values $< v_1, id_1 > \ldots < v_k, id_k >$ and string gaps $g_1 \ldots g_{k-1}$ which satisfy the following properties: (1) We assume that each wavelet coefficient in the string belongs to detail order O. Thus, this reflects the level of detail or granularity of each wavelet coefficient in the pattern. (2) The value v_i is an interval number from 1 through ϕ. (3) The value id_i is a number from 1 through l corresponding to the particular alphabet σ_{id_i} from Σ which the wavelet coefficient belongs to. (3) The value g_i is the gap between the end of the interval for $< v_i, id_i >$ and the beginning of the interval for $< v_{i+1}, id_{i+1} >$ in terms of the number of unit interval lengths of detail order O.*

We note that the above definition ensures that each element of the compositional pattern is derived from the same level of granularity of the wavelet decomposition.

Although the above definition of compositional pattern assumes uniform granularity, this is not the case for the discretized wavelet representation W of a given string in the original database. For ease of abstraction,[1] we will assume that each string is broken up into substrings of uniform granularity of the decomposition. Thus, for the ith string of length d_i, this will create $\log_2(d_i)$ such strings of differing levels of granularity. The string of level j is denoted by $W(j)$ for $j \in \{1 \ldots \log_2(d_i)\}$. Each such string $W(j)$ contains information corresponding to 2^{j-1} coefficients of order j for each of l symbols $\{\sigma_1 \ldots \sigma_l\}$. Thus, the length of $W(j)$ is $l * 2^{j-1}$. For further abstraction, we will break up the string $W(j)$ into the individual coefficients of each symbol σ_k, and denote this string by $W^k(j)$. This string $W^k(j)$ is an *ordered* sequence of the form $v_1 \ldots v_{2^{j-1}}$. Here, the value v_r is an integer from 1 through ϕ which corresponds to the discretized value of the rth wavelet coefficient of $W^k(j)$. We note that an interesting tradeoff exists between the ordering information and the compositional behavior retained in the strings of different orders. String wavelet representations of higher orders (high values of j in $W(j)$) retain greater ordering and *local* compositional information. On the other hand, strings of lower orders provide a better global overview of the variations in compositional behavior. Depending upon the particular data set, the classification characteristics of the strings could be hidden in either.

DEFINITION 2.2. *A compositional pattern $O < v_1, id_1 > g_1 < v_2, id_2 > g_2 \ldots g_{k-1} < v_k, id_k >$ is a substring of a given string W, if a sequence of positions $i, i+g_1 \ldots i+g_{k-1}$ can be found such that: (1) The ith position of $W^{id_1}(O)$ is v_1. (2) For each $q \in \{2 \ldots k\}$, the $i + g_{q-1}$th position in $W^{id_q}(O)$ is v_q.*

For a given level of detail, a compositional pattern is a substring of W if it shares similarity in the local compositional variations with W. Now, we will define the *support* of a compositional pattern.

DEFINITION 2.3. *The support of a compositional pattern is defined as the fraction of strings in the database which contain it as a substring.*

We note that not all compositional patterns of a particular support are equally important for the classification process. Compositional patterns which have high propensity to belong to particular class are distributed unevenly across the different classes. Therefore, we define the class confidence of a compositional pattern.

DEFINITION 2.4. *The confidence of a compositional pattern $\mathcal{P}$ for the class C_i is defined as the percentage of transactions containing the pattern $\mathcal{P}$ which also contain the class C_i.*

We shall henceforth refer to compositional patterns which have support s and confidence above the fraction c for any class as (c, s)-compositions. A (c, s)-composition $\mathcal{P}$ for the class C_i automatically induces the rule $\mathcal{P} \Rightarrow C_i$. Our aim is to find all the compositional patterns which induce the different classes in the data.

[1] In the actual implementation, only one string is maintained for efficiency. Our description eases understanding of the implementation.

The compositional patterns are generated using a two phase iterative process. In the first phase all the compositional patterns of length two are generated. The remaining compositional patterns are then generated iteratively in a level wise fashion. The overall process for generation of compositional patterns is illustrated in Figure 4. In order to generate patterns of length two, we use an iterative process in which we find patterns which have gaps starting from 0 to *maxgap*. For each particular value of the gap, this process is similar to that of finding 2-itemsets in databases. In this case, we however ensure that both the elements of the pattern are wavelet coefficients of the same order. Once such patterns of length two have been found, we use them in order to generate k-patterns by using an iterative methodology. In each iteration, we use joins in order to generate $(k+1)$-candidates from pairs of k-patterns. Let us assume that the set of all patterns of length k which have support at least s are denoted by $\mathcal{L}_k$. In order to decide whether a pair of patterns $\mathcal{P}_1 =< v_1^1, id_1^1 > g_1^1 \ldots g_{k-1}^1 < v_k^1, id_k^1 >$ and $\mathcal{P}_2 =< v_1^2, id_1^2 > g_1^2 \ldots g_{k-1}^2 < v_k^2, id_k^2 >$ are candidates for a join, they must satisfy the following properties:

- Both patterns must be of the same order.
- After removing the leftmost value and gap $< v_1^1, id_l^1 > g_1^1$ from $\mathcal{P}_1$ and the rightmost gap and value $g_{k-1}^2 < v_k^2, id_k^2 >$ from $\mathcal{P}_2$, the exact sequence of values and gaps in both the pruned patterns $\mathcal{P}_1'$ and $\mathcal{P}_2'$ are the same.

We shall denote this common segment in the two patterns as $\mathcal{P} = \mathcal{P}_1' = \mathcal{P}_2'$. Then, upon performing the join operation on the two patterns, we obtain the new pattern $< v_1^1, id_1^1 > \mathcal{P} < v_k^2, id_k^2 >$. All possible such $(k+1)$-candidates are then generated. We denote these candidate patterns by $\mathcal{C}_{k+1}$. We prune these patterns by using an analogous trick to that which is used by the *Apriori* method [2]. Specifically, all k-subset patterns of each member of $\mathcal{C}_{k+1}$ must be frequent and present in $\mathcal{L}_k$. Otherwise, it cannot have the required support and must be pruned from $\mathcal{C}_{k+1}$. All those patterns in $\mathcal{C}_{k+1}$ which have support greater than s are retained. This set of patterns $\mathcal{L}_{k+1}$ are the frequent $(k+1)$-patterns. This process continues in bottom up fashion until at some level, the set $\mathcal{L}_k$ is empty. Once all the frequent patterns are generated, we use them to generate the rules at the user-specified level of confidence. For each frequent pattern $P \in \mathcal{L}_i$ and class C_i, the rule $P \Rightarrow \mathcal{C}_i$ is generated is it has the desired minimum confidence c. We denote the final set of rules generated by $\mathcal{R}$. The formal pseudocode for the creation of rules is illustrated in Figure 4.

2.3 Rule Ordering and Pruning

Once these rules have been generated, we need to use them in order to actually classify the records. In order to do so, we find which rules are fired by a given test instance. In some cases, these rules may be conflicting, as a result of which it becomes necessary to develop precedence criteria. The criteria and algorithms developed for rule precedence and pruning share some common characteristics with those developed in [21] for the multidimensional classification problem. Given two rules R_1 and R_2, the rule R_1 has higher precedence than R_2 if:

- The confidence of R_1 is greater than that of R_2.
- In the event that the confidences are the same, the support of R_1 is greater than that of R_2.
- In the event that the supports and confidences are the same, the length of the compositional pattern for R_1 is lower than that of R_2.

We assume that the rules in $\mathcal{R}$ are sorted in the order of their precedence. In order to prune the rule set, we will analyze the *coverage behavior* of this rule set in conjunction with the precedence. A data point x is said to be *covered* by a rule, when the antecedent of the rule is a subpattern of the wavelet transformed representation of x. A data point is said to be *consistent* with a rule, when the class label in the consequent of that rule is the same as the label of the data point x.

We initialize the final rule set $\mathcal{F}$ to the null set $\{\}$. For each record x in the data set $\mathcal{D}$, we examine the rules in $\mathcal{R}$ in the order of precedence, starting at the highest. We find the first rule R' from $\mathcal{R}$ which covers x. Then, we check whether R' is consistent with x. If so, then it is marked. Otherwise, the data point x is removed from $\mathcal{D}$. We repeat this process for each of the data points. At the end of this procedure, a truncated database remains, along with the rules in $\mathcal{R}$ which have now been marked. We retain only the marked rules from $\mathcal{R}$ in the final rule set, while maintaining their original order of precedence. The majority class in the truncated data set is denoted as *defaultclass*. The ordered set of rules together with the default class form the model which is used for the classification of individual test instances. The procedure for the reordering and pruning of rules is illustrated in Figure 5.

Once these rules have been generated, the classification of test instances is relatively straightforward. The order of precedence of the rule set along with the *defaultclass* provides a classification algorithm. For each test instance T, we first determine its wavelet transformed representation. We used the transformed representation to determine the highest precedence rule which covers the test instance T. The class label in the consequent of this rule is reported as the class of T. If no rule covers the test instance T, then *defaultclass* is reported as the class label. The formal pseudocode for the classification procedure is illustrated in Figure 6.

3. EMPIRICAL RESULTS

We tested the system on an AIX 4.1.4 system with 200 MB of main memory. We generated two different kinds of data sets in order to demonstrate the effectiveness of the method over a wide range of problems. We tested the algorithm with two different types of data sets:

- The mouse data set: We generated this data set by querying the Entrez database. The strings were nucleotide sequences from the mouse genome. We generated a total of 173 sequences[2], each of which was labeled as *intron* or *exon*. Each pattern in the database was created from the four symbols $\Sigma = \{A, C, T, G\}$. We denote this data set as MGS (Mouse Genome Set). In addition, in order to test the reliability of the system to incomplete sequences, we created fragmented sequences by sampling subsequences from each pattern. Each element was included or excluded from its

[2]We note that this is a small subset of the total number of such patterns present in the Entrez database.

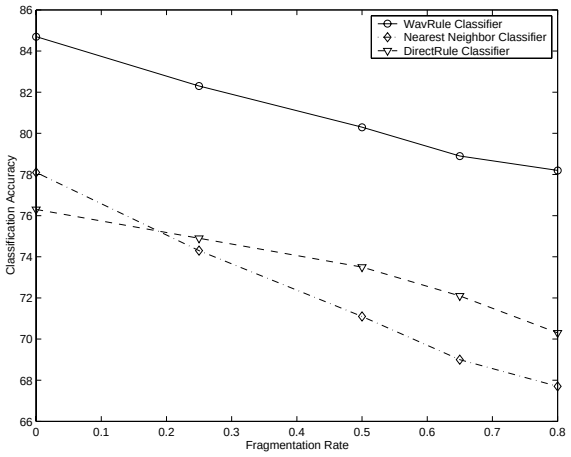

Figure 7: Effects of increasing fragmentation on Mouse Genome Data Set

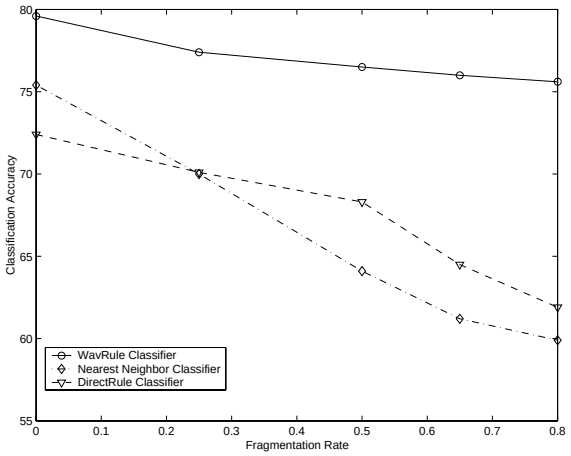

Figure 8: Effects of increasing fragmentation on Web Access Data Set (WAS1)

base pattern with equal probability of 0.5, and the order was maintained.

- The web access data set: This data set was created by sampling web pages from a proxy traces. Each sequence was drawn from the alphabet $\Sigma = \{ibm, other\}$ depending upon the whether the page accessed belonged to the ibm.com domain or not. The sequences were labeled *morning* or *evening*, depending upon the time of access of the first web page in the sequence. We had two traces from which we generated two data sets which we will henceforth refer to as WAS1 and WAS2 respectively. These data sets contained 521 and 644 sequences respectively. As in the case of the mouse data set, we created fragmented version of the data sets, which we refer to as WAS1-F and WAS2-F respectively.

As a baseline for effectiveness, we tested the nearest neighbor classifier using global alignment [4]. In addition, in order to test the effectiveness of the wavelet decomposition itself, we used a classifier which used exactly the same algorithm as discussed above except that it used the raw strings

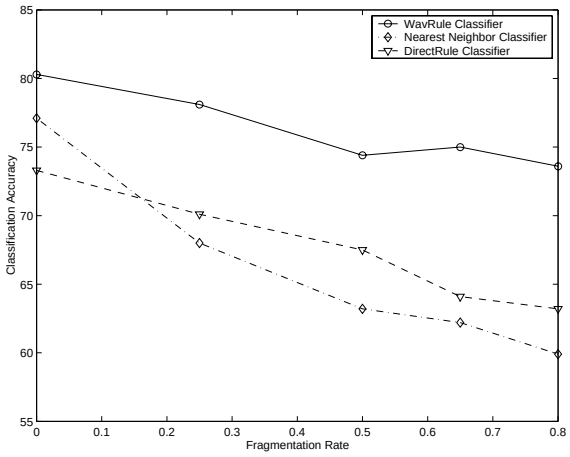

Figure 9: Effects of increasing fragmentation on Web Access Data Set (WAS2)

rather than the transformed database in order to generate the patterns. Thus, in this case, each generated patterns was of the form $s_1 g_1 s_2 g_2 \ldots g_{k-1} s_k$. Here, each s_i belongs to $\Sigma = \{\sigma_1 \ldots \sigma_l\}$ and g_i is a gap of length equal to 0 or more alphabets. We note that the only difference is that in this case, there is no concept of the "order" of a pattern. Thus, the same rule generation algorithm was implemented, except for the difference in how joins were computed in order to determine the candidate patterns. We will henceforth refer to this algorithm as the *DirectRule* method.

In Table 2 we have illustrated the results obtained by using the different methods on the six data sets. For the *WavRule* method, each string was truncated to the nearest power of 2 in order to facilitate the implementation of the wavelet decomposition. In order to account the possible effects of information loss from the truncation, we used the original string was used for the other methods. In each case, we set the support so as to mine the top 1000 patterns from the data. At the same time, a confidence threshold of 90% was used in order to determine the actual rules. It is clear that in each case, the *WavRule* method was significantly more effective than both the nearest neighbor and *DirectRule* classifiers. The nearest neighbor classifier performed quite poorly for the case of the web access data sets as compared to the mouse genome data sets. In both the mouse genome and the web access data sets, the classification accuracy of the nearest neighbor classifier was significantly worse for the fragmented data set as compared to the unfragmented data. It is interesting to note that in both cases, the nearest neighbor classifier was better than the *DirectRule* method only for the unfragmented versions. The reason for this was that the modified data set reflected only some of the substring fragments from the data. A nearest neighbor technique was mislead to a greater extent by the fragmentation process because it reduced the reliability of the global objective function significantly. On the other hand, both the *WavRule* and *DirectRule* classifiers were able to identify local subpatterns which were relatively unaffected by the fragmentation. In both cases, the *DirectRule* classifier was not as accurate as the nearest neighbor classifier before the fragmentation, but the relative performance of the two classifiers was reversed after the fragmentation. On

Table 2: Classification Accuracy Results

Data Set	WavRule Accuracy	NN Classifier	DirectRule Accuracy
MGS	84.7%	78.1%	76.3%
MGS-F	80.3%	71.1%	73.5%
WAS1	79.6%	75.4%	72.4%
WAS1-F	76.5%	64.1%	68.3%
WAS2	80.3%	74.4%	73.3%
WAS2-F	77.1%	63.2%	67.5%

Table 3: Efficiency Results

Data Set	WavRule Classification Speed/Record	NN Classification Speed/Record	WavRule (Training Time)
MGS	0.05 sec	8.2 sec	120.3 sec
MGS-F	0.03 sec	2.9 sec	89.9 sec
WAS1	0.07 sec	1.3 sec	35.3 sec
WAS1-F	0.05 sec	0.5 sec	20.4 sec
WAS2	0.06 sec	1.4 sec	44.7 sec
WAS2-F	0.04 sec	0.9 sec	33.3 sec

Table 4: Statistics of Patterns Found

Data Set	Average Pattern Length	Average Pattern Order	Pattern Order Variance
MGS	3.1	2.5	1.3
MGS-F	2.7	2.3	1.4
WAS1	2.6	2.3	1.2
WAS1-F	2.2	2.4	1.5
WAS2	2.5	2.4	1.6
WAS2-F	2.1	2.3	1.7

the other hand, the *DirectRule* method did not use the advantages of the Wavelet representation in identifying useful compositional characteristics of the data. As a result, in each case it could not match the effectiveness of the *WavRule* classifier in spite of its similarities to the latter in all other respects. In Table 4, we have illustrated the statistics of the average pattern length and order in the antecedent of the rules which was used to classify the test instances. While the pattern lengths are relatively short, they are often of varying orders. This tends to indicate two facts: (1) The classification behavior was hidden in small local regions of the string. (2) The classification behavior is often created by compositional behavior at different levels of granularity, since the most discriminatory patterns vary considerably in order. We note that neither the nearest neighbor classifier nor the *DirectRule* method is capable of providing such insight. In particular, the nearest neighbor classifier was unable to achieve either of the above two goals.

In order to test the robustness of the *WavRule* classifier further, we tested the data at varying levels of fragmentation. We define f as the fraction of the symbols which are sampled from the string in order to create the subsequences of these strings. In Figure 7, we have illustrated the accuracy of all classifiers with increasing fragmentation rate of the strings for the mouse data set. It is clear that with increasing fragmentation rate, the *WavRule* classifier outperforms the nearest neighbor classifier by an increasing margin. The differences are particularly pronounced in the case of the web data sets as illustrated in Figures 8 and 9. In all cases, even the *DirectRule* classifier performed better than the nearest neighbor classifier at higher fragmentation rates. The reason for the poor performance of the nearest neighbor classifier was that the lengths of the sequences were quite small in the case of the web data set. Small amounts of fragmentation were able to change the order of the distances sufficiently so that the methodology was no longer effective. This confirms the fact that the *WavRule* method is significantly more robust than the nearest neighbor classifier. The reason for this is its combination of a compositional and positional technique for classification which continues to retain its robustness over different kinds of data sets.

3.1 Efficiency of the Classification Process

We note that the nearest neighbor classifier is somewhat cumbersome because it requires the calculation of the edit distance between the strings from scratch. Since this distance calculation requires a dynamic programming algorithm, the time complexity of the most efficient algorithm is worse than quadratic in the average pattern length. Furthermore, because of the lack of suitable nearest neighbor indexes for complex distance functions such as the edit distance, the nearest neighbor algorithm needs to compute this value over all strings in the database sequentially. This leads to a database size dependent time for testing. The tradeoffs in the case of the *WavRule* classifier are somewhat different. While the *WavRule* classifier is extremely efficient in classification of test instances by several orders of magnitude, it requires an additional time for training which is not required by the nearest neighbor classifier. However, we note that this training time cost is incurred only once, after which the model can be efficiently used for any number of test instances. In Table 3, we have illustrated the running time for classification of individual test instances for each of the two methods. The classification time per record was determined by averaging the speed over a set of 100 records. In each case, the classification process of the *WavRule* method was two orders of magnitude faster. The differences were particularly significant in the mouse genome data set because of the fact that the pattern lengths were much longer. For the same reason, the nearest neighbor classification algorithm was much more inefficient on the unfragmented versions of the data sets. We note that these differences are in spite of the small database sizes. For larger training databases, the nearest neighbor classifier would show increasing classification time, whereas the *WavRule* method would continue to show similar efficiency per test instance. When the number of test instances are also large, then the nearest neighbor classifier would become an unattractive option. Table 3 also illustrates the training time of the *WavRule* classifier. About 98% of this time was utilized in the pattern generation process. The underlying training algorithm scales linearly with database size because of the natural database scan-based implementation. While this time cannot be directly compared to the instance-specific testing time of the nearest neighbor classifier, it is clear that the time per training instance is also significantly lower than the classification time of the nearest neighbor classifier.

4. CONCLUSIONS AND SUMMARY

In this paper, we discussed a technique for effectively leveraging the advantages of the wavelet representation technique in order to construct an effective classifier for strings. The beauty of string centered wavelet decomposition is that it can be combined with rule based methods in order to use both the composition and the ordering information at varying levels of locality and granularity. As a result, strings of varying lengths can be classified effectively using this method. We illustrated the advantages of our wavelet based classifier over the nearest neighbor classification method.

5. REFERENCES

[1] R. Agrawal, K.-I. Lin, H. Sawhney, K. Shim. Fast Similarity Search in the presence of noise, scaling, and translation in time series databases. *VLDB Conference*, 1995.

[2] R. Agrawal, R. Srikant. Fast Algorithms for finding association rules. *VLDB Conference*, 1994.

[3] R. Agrawal, R. Srikant. Mining Sequential Patterns. *ICDE Conference*, 1995.

[4] M. Deshpande, G. Karypis. Evaluation of Techniques for Classifying Biological Sequences. *Technical report, TR 01-33, University of Minnesota*, 2001.

[5] R. Duda, P. Hart. *Pattern Analysis and scene analysis*, Wiley 1973.

[6] J. Gehrke, V. Ganti, R. Ramakrishnan, W.-Y. Loh. BOAT: Optimistic Decision Tree Construction. *SIGMOD Conference*, 1999.

[7] J. Gehrke, R. Ramakrishnan, V. Ganti. Rainforest- A Framework for Fast Decision Tree Construction of Large Data Sets. *VLDB Conference*, 1998.

[8] J, Gehrke, W.-Y. Loh, R. Ramakrishnan. Data Mining with Decision Trees. *ACM SIGKDD Conference Tutorial*, 1999.

[9] V. Guralnik, G. Karypis. A Scalable Algorithm for Clustering Sequential Data. *ICDM Conference*, 2001.

[10] V. Guralnik, J. Srivastava. Event detection from time series data. *KDD Conference*, 1999.

[11] D. Gusfield. Algorithms on Strings, Trees and Sequences. *Press Syndicate of the University of Cambridge*, 1997.

[12] J. Han, G. Dong, Y. Yin. Efficient Mining of partial periodic patterns in time series databases. *ICDE Conference*, 1999.

[13] H. Jagadish, N. Koudas, S. Muthukrishnan. Mining Deviants in a Time Series Database. *VLDB Conference*, 1999.

[14] H. Jagadish, N. Koudas, S. Muthukrishnan. On Effective Multidimensional Indexing of Strings. *SIGMOD Conference*, 2000.

[15] M. James. Classification Algorithms, *Wiley*, 1985.

[16] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[17] D. A. Keim, M. Heczko. Wavelets and their Applications in Databases. *ICDE Conference*, 2001.

[18] E. J. Keogh, M. J. Pazzini. An enhanced representation of time series data which allows fast and accurate classification, clustering and relevance feedback. *KDD Conference*, 1998.

[19] E. Keogh, P. Smyth. A probabilistic approach to pattern matching in time-series databases. *KDD Conference*, 1997.

[20] E. Keogh, K. Chakrabarti, S. Mehrotra, M. Pazzini. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *SIGMOD Conference*, 2001.

[21] B. Liu, W. Hsu, Y. Ma. Integrating Classification and Association Rule Mining. *KDD Conference*, 1998.

[22] S. Manganaris. Learning to Classify Sensor Data. *TR-CS-95-10*, Vanderbilt University, March 1995.

[23] T. Oates. Identifying distinctive subsequences in multivariate time series by clustering. *KDD Conference*, 1999.

[24] C. Perng, H. Wang, S. Zhang, S. Parker. Landmarks: A new model for similarity-based pattern querying in time-series databases, *ICDE Conference*, 2000.